(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
22 February 2001 (22.02.2001)

PCT

(10) International Publication Number
WO 01/13303 A1

(51) International Patent Classification[7]: G06F 17/60

(21) International Application Number: PCT/US00/22318

(22) International Filing Date: 15 August 2000 (15.08.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/148,756 16 August 1999 (16.08.1999) US
09/437,098 9 November 1999 (09.11.1999) US

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier application:
US 60/148,756 (CON)
Filed on 16 August 1999 (16.08.1999)

(71) Applicant (for all designated States except US): VERSATA, INC. [US/US]; 2101 Webster Street, 8th Floor, Oakland, CA 94612 (US).

(72) Inventors; and
(75) Inventors/Applicants (for US only): HUBER, Val, J. [US/US]; 621 Santa Rosa, Berkeley, CA 94707 (US). MCKINNEY, Denny, G. [US/US]; 32 Jewell Street, San Rafael, CA 94901 (US). PATEL, Rahul [IN/US]; 6608 Crestwood Drive, Castro Valley, CA 94552 (US). SINGH, Vinod [IN/US]; 5488 Moonlight Common, Fremont, CA 94555 (US).

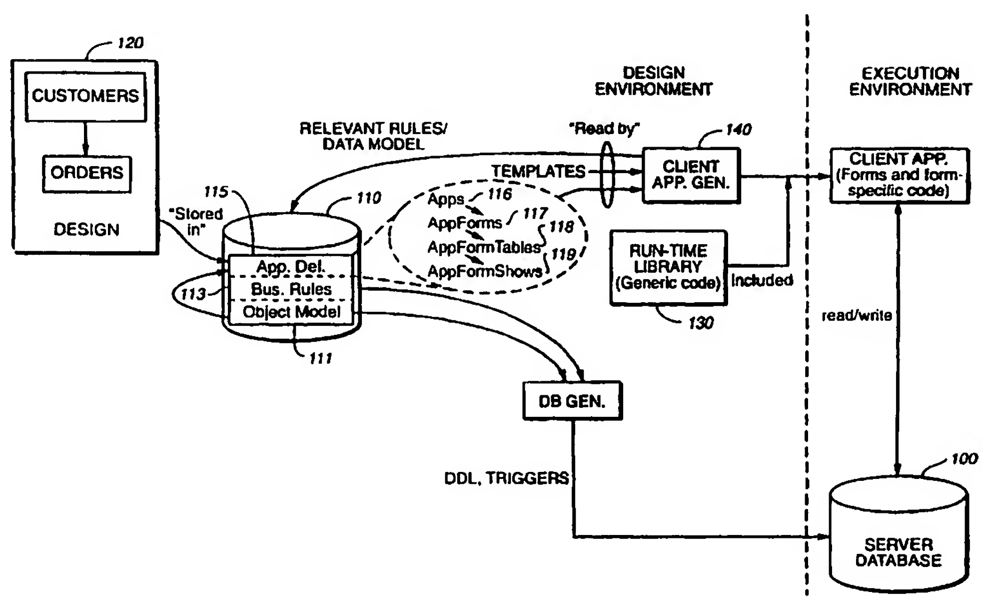
(74) Agent: KREBS, Robert, E.; Burns, Doane, Swecker & Mathis, LLP, P.O. Box 1404, Alexandria, VA 22313-1404 (US).

(81) Designated States (national): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States (regional): ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

[Continued on next page]

(54) Title: BUSINESS RULES AUTOMATION IN DATABASE APPLICATION DEVELOPMENT AND MAINTENANCE



(57) Abstract: The present invention, generally speaking, provides a data application development method for allowing a user to specify business rules (113) and application definition information (115). The application information (115) may be captured as the user draws the application within an application drawing area (120).

WO 01/13303 A1

Published:
— *With international search report.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

BUSINESS RULES AUTOMATION IN DATABASE APPLICATION DEVELOPMENT AND MAINTENANCE

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to database application development.

2. State of the Art

Database application software is layered on top of database software provided by such vendors as Oracle, Sybase, Informix, etc. The database application software enables the underlying database software to be used by non-experts to perform everyday business tasks, e.g., order entry, invoicing, collections, etc. Database application development is typically an arduous, time-consuming process performed by skilled computer programmers, and database application maintenance is often more arduous still.

Software tools have been developed to automate (to a greater or lesser degree) various aspects of database application development, including data definition and user interface definition. Data definition involves describing the structure of information in a database. A Data Definition Language (DDL) is a specialized language used to create new databases and specify the logical structure of the data (i.e., the database schema). One notation for representing database schemas is the "entity-relationship" (E/R) model, which is graphical in nature and uses boxes and arrows to represent the essential data elements and their relationships.

For data definition, CASE (Computer Aided Software Engineering) tools are available that generate one or more DDL files from E/R diagrams. For user interface definition, so-called "4GL" and "screen painter" products are available that automate the binding of tables to user interface screens with basic services for

both retrieval and update. These activities, however, represent only a fraction of the work involved in database application development.

Much (or even most) of the work involved in database application development relates to business rules. Business rules automate data manipulation. An example of a business rule is "the customer balance is the sum of the unpaid orders" Typically, business rules are implemented using triggers and stored procedures. A stored procedure is a callable program that is declared as an object within a database schema. A trigger is a stored procedure invoked in response to an insert, update or delete SQL (Structured Query Language) statement. Triggers and stored procedures are manually coded by skilled programmers. Such coding can be a painstaking, error-prone process.

Alternatively, such logic can be added into Applications Servers, for example as EJB (Enterprise JavaBeans) objects. This provides various advantages and scalability (one can provide multiple application servers to balance the load), or integration (since the data is no longer restricted to SQL). However, the underlying problem remains: coding an EJB is just as hard/time-consuming as coding a trigger. Both suffer from a single underlying root cause: reliance on procedural programming languages to enforce business logic.

Furthermore, business rules can and should evolve with the business. Changing business rules calls for code rework, also a painstaking, error-prone process. The distinct possibility (or probability) is that in a fast-changing business environment, without the expenditure of exorbitant amount of resources, information technology will be unable to keep up with the business.

What is needed, then, is a database application development tool that allows business rules to be specified in manner that is readily understood and that automatically builds or constructs code to implement and enforce those business rules.

SUMMARY OF THE INVENTION

The present invention, generally speaking, provides a data application development method for allowing a user to specify business rules (data requirements pertaining to one or more business functions) and for automatically generating code that enforces those business rules. As in a spreadsheet, rules may be readily specified and modified without concern for interactions between various rules. A business rules compiler performs rule/transaction mapping, rules ordering, rules optimization, and rules targeting to automatically generate code that enforces the business rules.

BRIEF DESCRIPTION OF THE DRAWING

The present invention may be further understood from the following description in conjunction with the appended drawing. In the drawing:

Figure 1 is a diagram illustrating the process of developing a database application in accordance with two-tier deployment;

Figure 2 is a diagram illustrating the process of developing a database application in accordance with three-tier deployment;

Figure 3 is a screen display of a Business Rules Designer, including a Derivation tab;

Figure 4 is a screen display of a Constraints tab of the Business Rules Designer;

Figure 5 is a screen display of a Relationships tab of the Business Rules Designer;

Figure 6 is a screen display of an Events tab of the Business Rules Designer;

Figure 7 is a screen display of a Rules Builder;

Figure 8 is a partial E/R diagram useful in illustrating rule chaining; and

Figure 9 is a block diagram of a Business Rules Compiler.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

In the following description, the overall database application development environment will be described, followed by the manner in which business rules are specified in accordance with an exemplary embodiment of the invention. The manner in which business rules are compiled will then be described.

Development Environment

An overview of the development process using the present development tool is shown in Figure 1. Referring to Figure 1, a design-time environment is shown on the left-hand side of the figure, and a run-time environment is shown on the right-hand side of the figure. In order to generate a multi-form application to run against a database server 100, a local database, or repository 110, is created. Within the repository, a data model 111 of the server database is created. To this data model is added additional "layers" of information, including business rules 113 (described in detail hereafter) and application definition information 115. The application definition information 115 may be captured as the user draws the application within an application drawing area 120 (displayed on a computer screen, not shown) using drag-and-drop techniques as described more fully in U.S. Application Serial No. 08/630,020 entitled AUTOMATED CLIENT/SERVER DEVELOPMENT TOOL USING DRAG-AND-DROP METAPHOR, filed April 9, 1996, incorporated herein by reference.

The application definition information 115 may itself be in the logical form of a database. In a preferred embodiment, an application is described as a collection of table rows related in one-to-many fashion. An application describes a multi-form program and is represented by a table row within a table JadeApps 116. ("jade" stands for "Java Application Development Environment.") A database may contain several applications, each of which is represented by an

application drawing (120). An application may contain many forms, each of which is described by a single row within a table JadeAppForms 117. A form may contain multiple tables, each of which is described by a table row within a table JadeAppFormTbls 118. Finally, each table may respond to user input to cause a transition to another form. Each transition is described by a table row within a table JadeAppFormShows 119.

The present development tool may be built upon and leverage the capabilities of existing popular software packages. In an exemplary embodiment, Microsoft Access is used as the repository. Microsoft Access is a relational database management system that is part of the Microsoft Office Pro application suite. Of course, other database management systems and other target 4GL programming languages may be used.

Generated applications can take a number of organizations. In a "code generation" approach, an application contains two types of code: form-specific code and generic code. Code that pertains to a particular form is form-specific code. Other code (stored as "modules") is more general in nature. In the case of the present tool, this includes, for example generic code that, when a row becomes current, synchronizes each Dependent Recordsource (i.e., opens Dependent Recordsources using parameter values from the source Recordsource). Referring still to Figure 1, the generic code constitutes a run-time library 130 (Vision Foundation Classes, or VFC) that is "included" as part of the final application.

Or, generated applications can be a set of classes, one for each form. Each such form class includes a set of objects corresponding to both graphical objects (text boxes, grids and so forth) and semantic objects that provide behavior to retrieve/update data, perform form transitions, or pick foreign key values. The set of objects in conjunction with their property settings (set in the Design Environment) provide the desired behavior. In the present system (Vision Jade),

this behavior is inherited from a set of VFC classes (of which objects are instances) that operate (per property settings) to provide the designated behavior.

Both organizational approaches are, for this purpose, equivalent.

The database code generator creates data objects within the server database based on information stored in the data model using, for example, DDL (Data Description Language). In addition, in the case of a two-tier implementation, the database generation code generates trigger code that enforces the business rules specified in the repository. Because no manual coding is required, either on the client side or the server side, the application can be easily modified, repeatedly if necessary, and regenerated. The "coding waterfall" problem characteristic of the prior art is thus avoided.

Referring to Figure 3, in the case of a three-tier implementation, the process is similar. However, instead of the client application interacting directly with the database, an additional server tier, the "business logic server," is introduced. The business logic server includes business objects--classes that enforce business rules--and application server code. The application server code performs such task as load balancing, fail over, communications polling, and physical read/write transactions with the database. The database stores data objects--the physical data itself. Data represented by data objects (also be referred to as base components) physically reside on disk. Also of interest are query objects--SQL constructs that allow the user to, from the point of view of the user, define combination data objects, e.g., joins/projections of data objects. Query objects allow for the realization of "virtual" business objects. Data objects and query objects are subclasses of business objects within the object model hierarchy followed by one embodiment of the invention.

Business logic server code is generated by a business logic server code generator based on the business rules specified in the repository. These may, for example, operate as Corba or EJB objects, or some equivalent distributed object

protocol. The database code generator no longer generates trigger code as in the two-tier model.

In general, the present invention is applicable to an "N-tier" architecture. The business logic server may, for example, interface to a separate Web server. More than one business logic server may be provided for scalability reasons. Multiple business logic servers may be provided, each one implementing a particular business function, e.g., payroll, inventory, etc.

Business Rules--Specification

Declarative rules provide a simple mechanism to enforce business requirements across business functions. A business function is an action performed by the user of an application to fulfill a specific business purpose, for example, the entry of a new order. A business requirement is a statement for objectives for part of a business function that identifies the objectives for working with data. A requirement usually encompasses a series of validations and computations that begin as an update on a data object and may in turn affect other data objects. For example, the entry of each order item in a new order could include the requirement that the cost of the order item must not cause the customer balance to exceed the customer's credit limit.

The functionality of declarative business rules is comparable to spreadsheet functionality as seen in Table 1.

Table 1

Spreadsheet functionality	Declarative business rule functionality
The formula to determine a spreadsheet cell value may refer to many other spreadsheet cell values, each determined by its own formula.	The rule to determine a column value may refer to many other column values, each determined by its own rule.
A change to the value or the formula for a spreadsheet cell, or the insertion or deletion of spreadsheet rows, may cause automatic changes to many other cell values that refer to the changed cell value in their own formulas.	A change to the value or rule for a column, or insertion or deletion of records, may cause automatic change to many other column values that refer to the changed column value in their own rules. Each declarative business rule is defined in a single data object. Because cascading rules automatically trigger other rules, business rules can be combined to implement multi-data object update processing.
A change to a single cell in a set of linked spreadsheets may cause changes to cells in many other spreadsheets.	A change to one business rule may affect many business processes which are implemented through that rule.

A further important comparison may be drawn between spreadsheet functionality and business rule functionality. In a spreadsheet, changing a formula is straight-forward. Similarly, because the business rules compiler described hereafter automates rules processing and addresses dependencies between business rules, modifying the business logic is a fairly short and simple procedure. Rules are unordered, so the user does not have to be concerned with ordering when entering (or, more importantly, altering) rules.

Various different types of business rules are defined. The most important types for purposes of the present description are derivation rules, constraint rules and referential integrity rules.

Derivation rules define how a column's value is computed when a database update occurs. Derivations can be aggregations of child record values (sums or counts--min., max., etc.), replicates (copies) of parent record values, or formulas based on values of other columns in the same record. A sum rule derives a parent column value by adding values of a specified column in related child data objects. A sum may be unconditional ("the total item amount is the sum of the individual item amounts") or conditional ("the customer balance is the sum of the *unpaid* orders," where unpaid is the condition). A count rule derives a parent column value by counting the number of records in related child data objects (e.g., the number of unpaid orders). Like sums, counts can be unconditional or conditional. Each derivation rule can cause other derivation rules to fire. That is, a change in dependent data may cause a derivation rule for "A" to fire, and since other derivations can be based on "A," a "chain reaction" occurs. This cascading of rules enables complex, multi-data object update processing.

Constraint rules ("constraints") are used to enforce conditions (boolean expressions) that refer to multiple columns for data validation. For example, a constraint may refer to an account balance column, ActBalance, and a credit limit column, CreditLimit. Whenever one of these columns is updated, the constraint then validates the result (e.g., the update is rejected when the ActBalance > CreditLimit). Constraints can refer to the results of derivation rules and can thus govern derived columns.

Referential integrity rules preserve relationships between data objects when updates occur. A referential integrity rule may prevent parent update if children are present, prevent parent delete if children are present, prevent child insert/update if parent is not present, etc. Various types of "cascade" referential

integrity rules may be specified, including cascade update (update child foreign keys on parent update of primary key), cascade delete (delete children on parent delete), cascade insert (insert parent if none on child insert/update), and cascade nullify (null child foreign keys on parent delete).

Business rules may be specified in various different ways. In an exemplary embodiment, a Business Rules Designer provides a graphical user interface to define business logic in the form of declarative business rules. Referring to Figure 3, the Business Rules Designer may consist of several overlapping tab sheets used to define different types of business rules, and a Rule Builder tool used for graphically building expressions.

In the embodiment of Figure 3, the Columns tab of the Business Rules Designer has a read-only grid of all the columns in a selected data object, with their column-level rule information. Rules are displayed within the grid but are not input directly into the grid.

Referring still to Figure 3, the Derivation tab allows a user to enter rules that define how a column's value is derived when update occur. Various options may be selected from the derivation-type box. Types of derivations include sums and counts, parent replicates, formulas and defaults (described above). For sums and counts, two buttons are available, a browser button that invokes the Rules Builder, described hereafter, and a syntax checker button.

Referring to Figure 4, the Constraints tab allows the user to define data object-level constraints that enforce multiple-column conditions for data validation. This tab provides a grid that lists information for all constraints defined for the selected data object. The Constraint Name field allows the user to specify a unique name for a constraint. The Condition field allows the user to enter an expression describing the constraint's condition, optionally using the Rule Builder.

Referring to Figure 5, the Relationships tab provides information about parent-child relationships for the selected data object. This tab also allows the user

to modify referential integrity rules. A relationships outline lists the parent and child relationships for the selected data object, and lists the primary and foreign keys for each relationship. The user selects a relationship from this outline to modify its rules. A referential integrity frame allows the user to modify the system's default referential integrity rules to preserve relationships between data objects when updates occur. An Enforce referential integrity checkbox provides separate sets of referential integrity rule options buttons for parent updates, parent deletes, and child insert/updates.

Referring to Figure 6, the Events tab allows the user to incorporate custom code into system-generated application code. A grid lists a name and description for each business rule event defined for a selected data object. Events may be of different types, e.g., an in-line trigger code event, a conditional action event, or a user-defined modification event. Of particular interest are conditional action events. A conditional action event call a specified action (function or stored procedure) to be executed when a defined condition evaluates to true. A Condition/Action box allows the user to enter an expression to specify the function or stored procedure to be executed and any arguments to be passed. A Rule Builder tool may be used to complete these fields if desired.

A Rule Builder screen display is shown in Figure 7. The Rule Builder is used to create expressions graphically. The Rule Builder provides lists of columns, functions, and keywords, as well as buttons with expression elements. The user clicks an item to include it in an expression. The list of columns uses outline controls for explorer-like expansion of column information. The list of functions includes built-in RDBMS functions and any custom functions such as stored procedures (2 tiered), or Java functions (3 tiered) that have been registered. The user, when entering conditional expressions for sum, count, column validation or constraint rules, etc., may select from the functions in the list box. The list of functions is extensible; i.e., the user may define new functions as methods in Java,

for example, and register those methods with an External Object Manager, after which they will be listed as functions in the list box for use within the Rules Builder.

The list of keywords includes Inserting, Updating, and ":Old". The :Old keyword allows the user to differentiate between a changed column value and its value before the transaction that caused the change.

Although a preferred method of specifying business rules has been described, various other methods may also be used, including simple text input using a text editor (a "rules language"), for example.

Defining business rules is basically the same whether a declarative approach or a procedural approach is followed, and involves identifying business functions and the requirements of those functions, then defining one or more rules that implement each requirement. The main difference between the two approaches is that, in the declarative approach, designing the business logic rule *is* the implementation; i.e., declaring the rules allows the code to be automatically implemented. In the procedural approach, such rules are, by contrast, merely an excellent specification for the code to be built by a programmer.

Business Rules -- Compilation

In an exemplary embodiment, the business rules implementation process has two important characteristics.

First, rules are unordered: the system automatically detects rule dependencies and computes an order of execution that is correct and optimal. Optimization is repeated when rules are changed, so performance remains high over maintenance changes. In the same manner, ordering analysis assures that rule enforcement remains correct (i.e., independent data is calculated first).

Second, rules are transaction-independent: the system automatically computes which transactions are affected by a rule and builds logic for those

transactions to implement the rule. For example, a rule regarding customer balance is automatically reused for many transactions, e.g., insert orders, delete orders, pay orders, etc.

Referring to Figure 9, a simplified block diagram is shown of a business rules compiler that automatically generates code to implement the foregoing business logic rules. Input to the business rules compiler may be in the form of a Business Rules Report, an example of which is shown in Appendix A.

Conceptually, the business rules compiler is divided into four stages, a rule/transaction mapping stage, a rule ordering stage, a rule optimization stage, and a rule targeting (code generation) stage. The operation of each of these stages will be described in turn.

The function of the rule/transaction mapping stage is to determine the conditions under which data changes must "chain" to fire dependent rules. In the simplest case, consider a Data Object/Order Item in which the attribute Amount has a Derivation Rule of Price times Quantity. The rules compiler must generate code to recompute Amount if a transaction occurs in which updates are made to Price or Quantity.

Consider a more complicated case where derivation rules reference attributes in other Data Objects. For example, the Customer Balance attribute's derivation depends on the Order's Amount and Paid attributes. As noted above, the system must detect transactions that alter an Order's Amount or Paid attribute, and recompute the Customer Balance. Moreover, the system must also detect transactions that *insert* Orders (increase the balance) or *delete* Orders (decrease the balance). While such transaction analysis may seem straightforward, it is a frequent source of error since the number of such cases to be analyzed is very large, and because changes to requirements result in subtle side-effects that render prior transaction analyses incorrect.

The rules ordering stage computes a correct order in which to perform the business logic operations identified during the rules mapping stage. A dependency graph algorithm is used for this purpose. The use of dependency graphs is well-known. Given a two-column table of derived and referenced attributes of the type described, the algorithm essentially looks for attributes that are referenced but are not themselves derived, i.e., attributes that appear on the left side of the table with no entries on the right side of the table. The corresponding derivation expression is then placed in order and removed wherever found from right side of the table, and the process is repeated. In this manner the table is progressively reduced.

For example, one derivation expression may be A = X*Y, and another derivation expression may be B = 5*A. Assuming for the moment that these are the only derivation expressions, then X and Y are referenced but are not themselves derived. Therefore, X and Y appear on the left side of the table with no entries in the right side of the table. "A" is referenced but is derived. Since X and Y have no entries on the right side of the table, they are removed from the right side of the table. As a consequence, A then has no entries on the right side of the table and is therefore removed from the right side of the table and code is generated to derive it. This leaves B with no entries (or rather only a constant entry) on the right side of the table so code is generated to derive it. Any instances of B (in this case there are none) are then removed from the right side of the table. The foregoing process defines an order in which the derivation expression A = X*Y is therefore placed before the derivation expression B = 5*A. The work of the ordering stage, in this example, is complete.

In some instances, there may be a mutual dependency, or ôcycle,ö that cannot be resolved. A cycle is detected when execution of a code loop performing the foregoing steps does not result in any removals, but the right side of the table is still not empty. A cycle is reported as an error to the user.

Together, the rules mapping and rules ordering stages identify operations and an execution order to achieve correct results in accordance with the rules. The optimization stage then computes an efficient implementation of these functionally correct operations. More particularly, Base Components (data objects physically residing on disk) provide insert(), update() and delete() methods which enforce Business Rules. This execution is highly optimized for good performance.

Note that many of the Business Rules stipulate cross-object processing, resulting in multi-object transactions. For example, changing an Order Item quantity requires the Part on-hand quantities to be adjusted and the Order Amount to be recalculated. The generated code performs automatic object caching and cache checking in recognition of the fact that a given change may require that more than one column of a data object be adjusted. When the data object is first read to allow the first data column to be adjusted, it is cached. The data object is not written back to main storage until later. Therefore, when a subsequent column is to be adjusted, the column of the cached copy is adjusted. Also, where multiple rules span the same relationship, the business rules compiler recognizes this and builds logic that enforces all these rules with a single I/O operation.

In general, the system buffers updates made during a transaction and re-uses these cached data for subsequent processing. Consider, for example, updates to two items within the same order. The first order update causes the order component to be retrieved and updated. Instead of saving the row to the database server, it is saved into cache. The second order update may therefore update the cached row without retrieving the order. The cached order row is the written to disk at the conclusion of Save All processing, when all rows have been processed.

Another optimization involves aggregation. As previously described, the customer balance may be defined as the sum of the unpaid orders. Assume that an order is changed, e.g., increased from 100 to 120. An inefficient implementation of the foregoing rule would be to, whenever an order is changed, re-sum all of the

unpaid orders. Instead, child objects generate adjustments to parent aggregate data in order to persistently maintain the aggregate data. Hence, when the order amount is increased from 100 to 120, 20 is added to the customer balance.

A further optimization involves rule by-pass. Because the business rules compiler analyzes all dependencies inferred by the rules, it can use this knowledge to build code that bypasses processing when underlying dependencies have not been affected. For example, altering an order's date requires no adjustment to the related customerÆs balance. Similarly, altering an orderÆs date requires no re-validation of an existing customer number (i.e., referential integrity).

Various other optimization may be performed.

At the completion of the optimization stage, the business rules compiler will have computed a correct and efficient order in which to enforce the business rules. The rule targeting stage then generates actual code to enforce the business rules. The code generation process may be better understood by considering what code needs to be generated for each of various kinds of rules, i.e., derivation rules, constraints, and referential integrity rules.

Recall that derivation rules may be based on formulas entered by the user or on aggregates or replicates. In the case of a user formula, the code generated is essentially of the form:

IF any-referenced-attribute-changed THEN do {calculate drivation formula}.

Observe now how these rules operate in a typical real world transaction. Table 2 depicts the Requirements (bold text) and resultant rules (bulleted points) as they might have been conceived by a developer and entered into the system. Imagine now that the order item Quantity is altered by a transaction. The circled numbers and arrows represent the rules that fire in response to the change, per the IF <reference fields changed> tests for each of the rules types described above.

Recall that the order of firing (represented by the arrows) is made correct by the Dependency Graph Analysis described above.

In the case of aggregations, the rules targeting stage generates code to, if a "hot" field (i.e., a field referred to in a derivation rule) changed, get the appropriate related record and add a "delta" to the aggregate, then make the object update itself. In the case of the example ôthe customer balance is the sum of the unpaid orders,ö for example, the paid flag and the order amount are "hot." If a paid flag is changed, then the customer balance is decreased by the corresponding order amount.

In the case of replicates, the rules targeting stage generates code to, if the replicate field is changed, copy a field to related fields. In the previous example, when an order is marked as paid, then order items belonging to that order may also be marked as paid.

In the case of constraints, the rules targeting stage generates code to check basic attributes. If a change satisfies a constraint, then the change is allowed. Otherwise, the change is not allowed, all partially complete updates are rolled back, and a message is returned to the client. Similarly in the case of referential integrity rules, the rules targeting stage generates code to check whether a change satisfies the rule. If a change satisfies the rule, then the change is allowed. Otherwise, the change is not allowed, and an error is signalled.

Consider, for example, the partial E/R diagram of Figure 8 in relation to the business function of inserting an order item. A business rule may enforce a customer credit limit, i.e., Customer Balance not > CreditLimit. Evaluation of this rules expression may in turn require that the Customer Balance be recomputed as the sum of unpaid Order Totals. A summary of various rules caused to fire by inserting an order item in a typical real-world example is given in Table 2.

Table 2

Check Credit Limit	
• Customer Balance not > CreditLimit	5
• Customer Balance = sum of unpaid Order Totals	4
Compute Order Total	
• Order Total = Freight + Amount Items	3
• Amount Items = sum (Item Amounts)	2
• Item Amount = QtyOrdered * Price	1
• Item Price = Part Price at time of order	
Determine whether Reorder required	
• Reorder if OnHand - QtyUnShipped < Reorder Level	7
• QtyUnShipped = sum(Item QtyOrdered) where unshipped	6
• Item.UnShipped is obtained from Order	

As described previously in relation to Figure 2, a three-tiered architecture using application servers allows data sources such as legacy (e.g., mainframe) applications and package solutions (e.g., PeopleSoft or SAP) to be used in conjunction with new data to support a custom business process. Current products provide basic capabilities to manipulate multiple data sources. Such products, however, are unable to integrate multiple data sources into a common data model together with new data, and to build *composite* applications using declarative business rules. This business requirement is met using an eXtensible Data Access (XDA) architecture that allows a user to define data objects whose data is not retrieved/written by SQL commands, but by any API (Application Program Interface). For example, a user may define data objects that are based on CICS APIs, PeopleSoft APIs, etc. Such data objects can co-exist in a repository describing data from multiple data sources, including local SQL data. In this manner, the repository integrates multiple disparate data sources into *a common data model.*

The declarative business rules methodology described heretofore may be extended to XDA data objects, allowing a user to define business rules on such data objects just as for SQL objects. One aspect of the XDA architecture is retrieval and another is update. When a client makes a retrieval request for an XDA object, the application server, instead of routing the request as normal to a default data manager, calls XDA code identified by the user in the repository definition of the Data object. For update, the application components, instead of emitting SQL, emit calls to user-supplied XDA code to read/write data.

To define an XDA data object, the user registers XDA code (e.g., a Java class) in the repository and develops the XDA code. The user then uses the Busness Rules Designer to designate this XDA Driver by specifying its name as a property value of desired Data Objects. Generally speaking, the XDA code is required to start a query, retrieve a row, and handle insert/update/delete operations.

It will be appreciated by those of ordinary skill in the art that the invention can be embodied in other specific forms without departing from the spirit or essential character thereof. The presently disclosed embodiments are therefore considered in all respects to be illustrative and not restrictive. The scope of the invention is indicated by the appended claims rather than the foregoing description, and all changes which come within the meaning and range of equivalents thereof are intended to be embraced therein.

What is claimed is:

1. A method of developing a database application, comprising:

specify business rules describing requirements pertaining to one or more business functions; and

compiling the business rules to automatically generate machine instructions for enforcing the business rules.

2. The method of Claim 1, wherein at least one of the business rules is a derivation rule, and specifying comprises forming a rules expression.

3. The method of Claim 2, wherein the rules expression is formed in accordance with a specified rules language syntax.

4. The method of Claim 2, wherein the rules expression is used in at least one of the following: sum/count qualification, formula evaluation, and constraint evaluation.

5. The method of Claim 2, wherein the rules expression is formed graphically by selecting at least one of an item from a list and a button labelled with a rules expression element.

6. The method of Claim 1, wherein at least one of the business rules is a constraint rule, and specifying comprises identifying multiple columns within the database.

7. The method of Claim 1, wherein at least one of the business rules is a referential integrity rule, and specifying comprises selecting from among multiple options a single option within each of multiple categories.

8. The method of Claim 1, wherein compiling the business rules comprises automatically identifying how changes to a changed field are to be propagated to one or more derived fields.

9. The method of Claim 8, wherein the changes are propagated to derived fields across multiple data objects.

10. The method of Claim 9, wherein a derived field is a qualified sum or count of a particular field in multiple related data objects.

11. The method of Claim 9, wherein a derived field is a replicate of a particular field in a related data object.

12. The method of Claim 11, wherein the derived field is maintained upon subsequent changes to the particular field in the related data object.

13. The method of Claim 1, wherein compiling the business rules comprises computing an order of execution of multiple processes that produces correct results in accordance with the business rules.

14. The method of Claim 13, wherein the multiple processes automate multiple inter-dependent rules.

15. The method of Claim 13, wherein compiling the business rules comprises automatically identifying optimizations for efficient execution of the multiple processes.

16. The method of Claim 1, wherein compiling the business rules comprises automatically generating code for enforcing the business rules.

17. The method of Claim 1, wherein the database application accesses SQL data.

18. The method of Claim 17, wherein the database application accesses SQL data and non-SQL data.

19. The method of Claim 1, wherein the database application accesses disparate types of data and enforces the business rules across the disparate types of data.

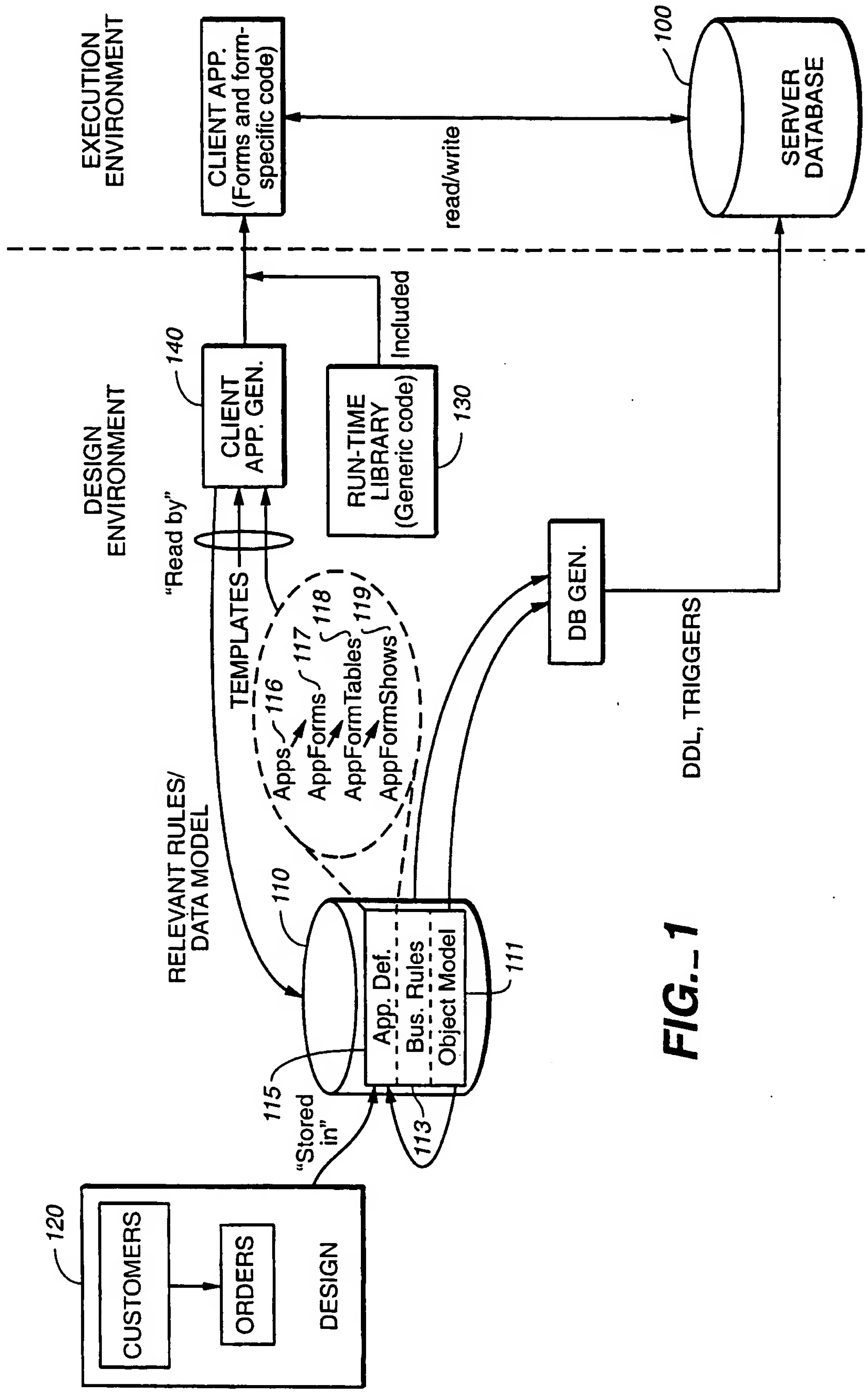

EXECUTION ENVIRONMENT
CLIENT APP. (Forms and form-specific code)
read/write
100
SERVER DATABASE
DESIGN ENVIRONMENT
140
CLIENT APP. GEN.
Included
RUN-TIME LIBRARY (Generic code)
130
"Read by"
TEMPLATES
116
117
118
119
Apps
AppForms
AppFormTables
AppFormShows
DB GEN.
DDL, TRIGGERS
RELEVANT RULES/ DATA MODEL
110
App. Def.
Bus. Rules
Object Model
111
115
113
"Stored in"
120
CUSTOMERS
ORDERS
DESIGN

FIG._1

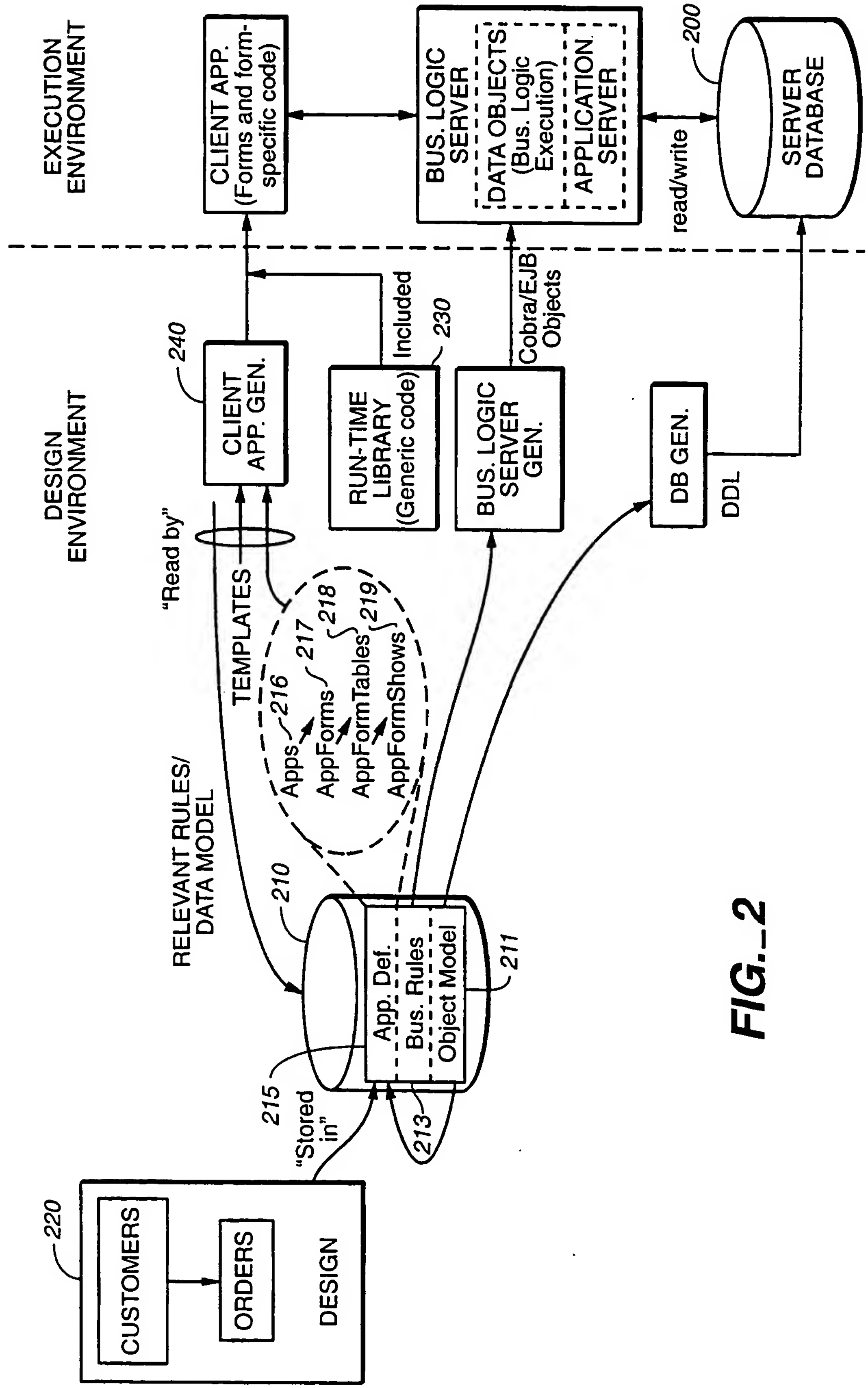
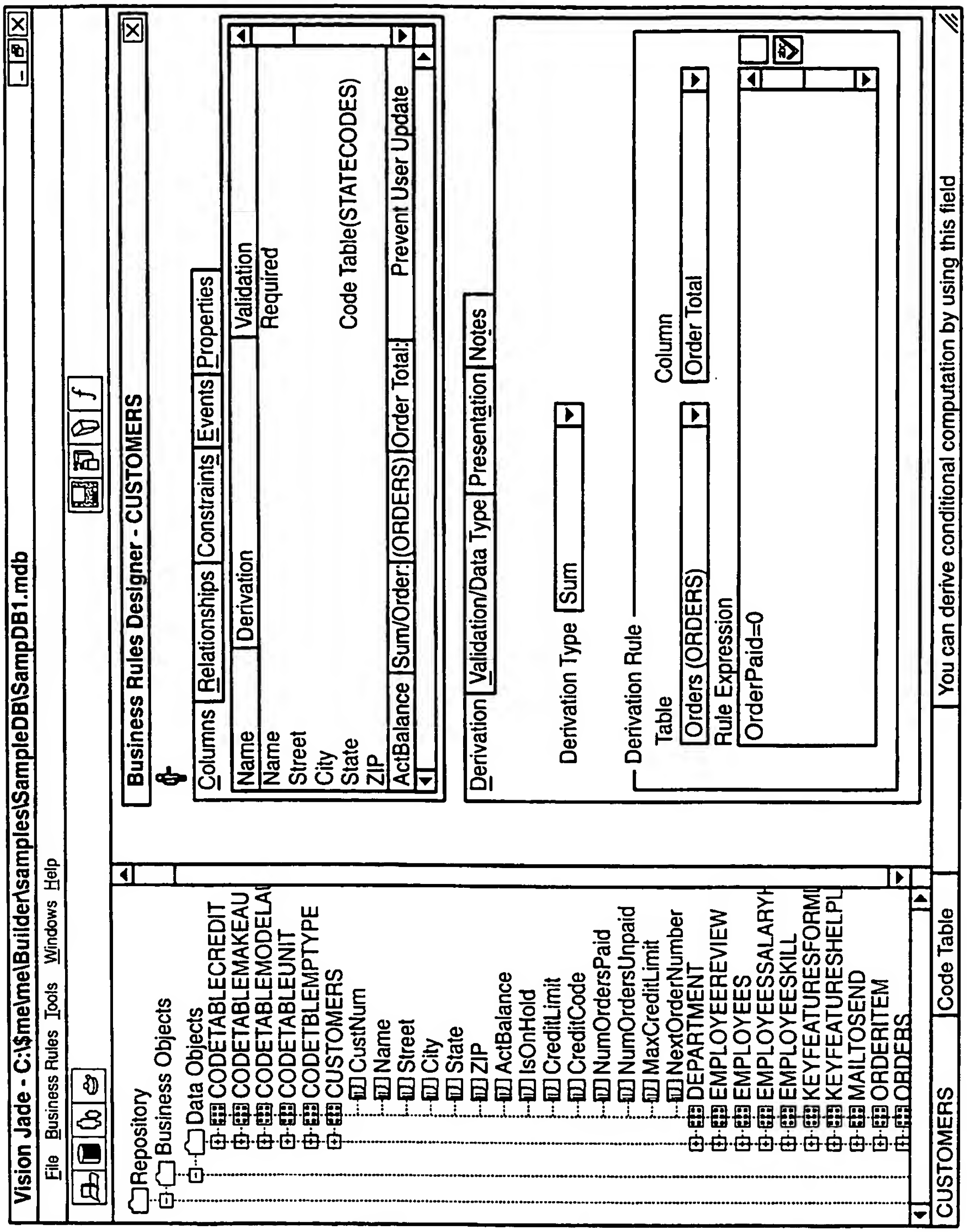

EXECUTION ENVIRONMENT
DESIGN ENVIRONMENT
CLIENT APP. (Forms and form-specific code)
BUS. LOGIC SERVER
DATA OBJECTS (Bus. Logic Execution)
APPLICATION SERVER
200
read/write
SERVER DATABASE
240
CLIENT APP. GEN.
Included
230
RUN-TIME LIBRARY (Generic code)
Cobra/EJB Objects
BUS. LOGIC SERVER GEN.
DB GEN.
DDL
"Read by"
TEMPLATES
216
217
218
219
Apps
AppForms
AppFormTables
AppFormShows
RELEVANT RULES/ DATA MODEL
210
App. Def.
Bus. Rules
Object Model
211
215
"Stored in"
213
220
CUSTOMERS
ORDERS
DESIGN

FIG._2



Vision Jade - C:\$melme\Builder\samples\SampleDB\SampDB1.mdb
File Business Rules Tools Windows Help
Repository
Business Objects
Data Objects
CODETABLECREDIT
CODETABLEMAKEAU
CODETABLEMODELA
CODETABLEUNIT
CODETBLEMPTYPE
CUSTOMERS
CustNum
Name
Street
City
State
ZIP
ActBalance
IsOnHold
CreditLimit
CreditCode
NumOrdersPaid
NumOrdersUnpaid
MaxCreditLimit
NextOrderNumber
DEPARTMENT
EMPLOYEEREVIEW
EMPLOYEES
EMPLOYEESSALARY
EMPLOYEESKILL
KEYFEATURESFORM
KEYFEATURESHELP
MAILTOSEND
ORDERITEM
ORDERS
Business Rules Designer - CUSTOMERS
Columns
Relationships
Constraints
Events
Properties
Name
Derivation
Validation
Name
Required
Street
City
State
Code Table(STATECODES)
ZIP
ActBalance
Sum/Order:(ORDERS) Order Total:
Prevent User Update
Derivation
Validation/Data Type
Presentation
Notes
Derivation Type
Sum
Derivation Rule
Table
Orders (ORDERS)
Column
Order Total
Rule Expression
OrderPaid=0
CUSTOMERS
Code Table
You can derive conditional computation by using this field

FIG._3

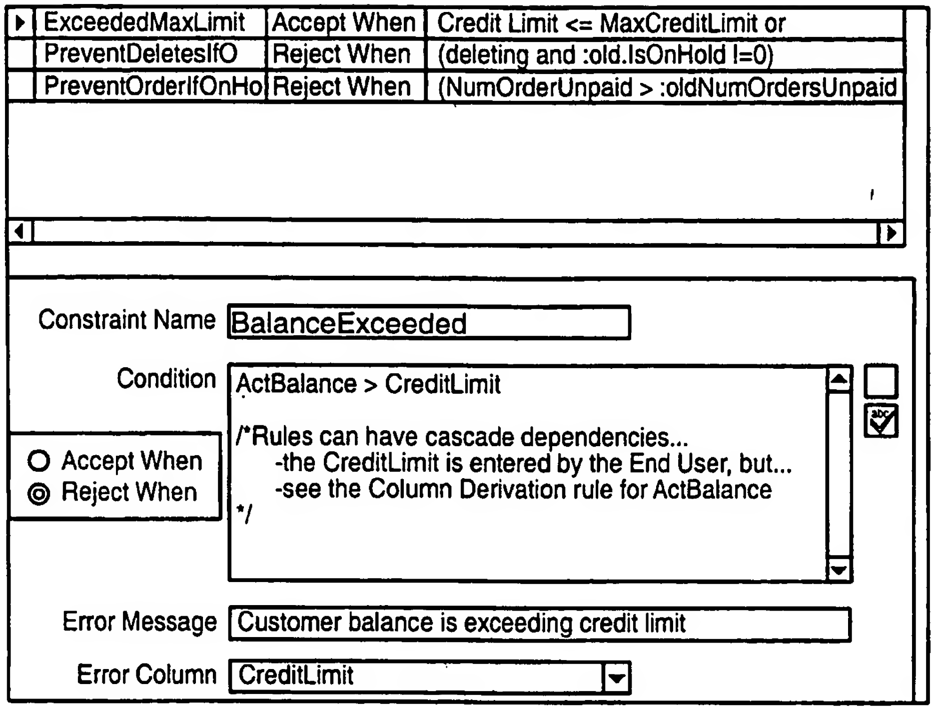

ExceededMaxLimit
Accept When
Credit Limit <= MaxCreditLimit or
PreventDeletesIfO
Reject When
(deleting and :old.IsOnHold !=0)
PreventOrderIfOnHo
Reject When
(NumOrderUnpaid > :oldNumOrdersUnpaid
Constraint Name
BalanceExceeded
Condition
ActBalance > CreditLimit
/*Rules can have cascade dependencies...
-the CreditLimit is entered by the End User, but...
-see the Column Derivation rule for ActBalance
*/
Accept When
Reject When
Error Message
Customer balance is exceeding credit limit
Error Column
CreditLimit

FIG._4

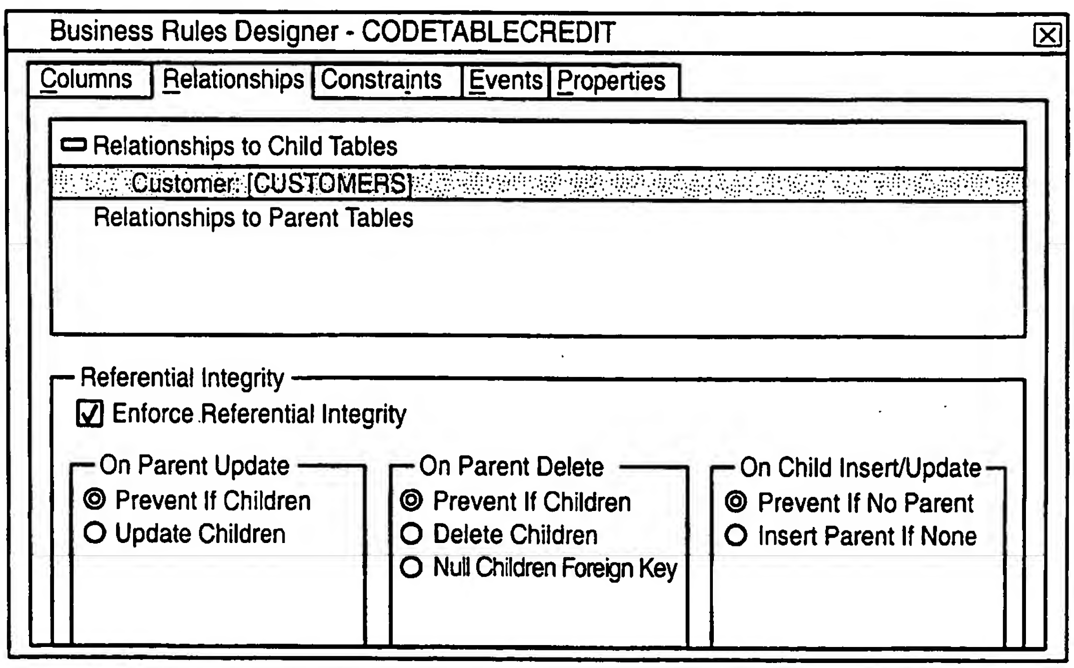

Business Rules Designer - CODETABLECREDIT
Columns
Relationships
Constraints
Events
Properties
Relationships to Child Tables
Customer: [CUSTOMERS]
Relationships to Parent Tables
Referential Integrity
Enforce Referential Integrity
On Parent Update
Prevent If Children
Update Children
On Parent Delete
Prevent If Children
Delete Children
Null Children Foreign Key
On Child Insert/Update
Prevent If No Parent
Insert Parent If None

FIG._5

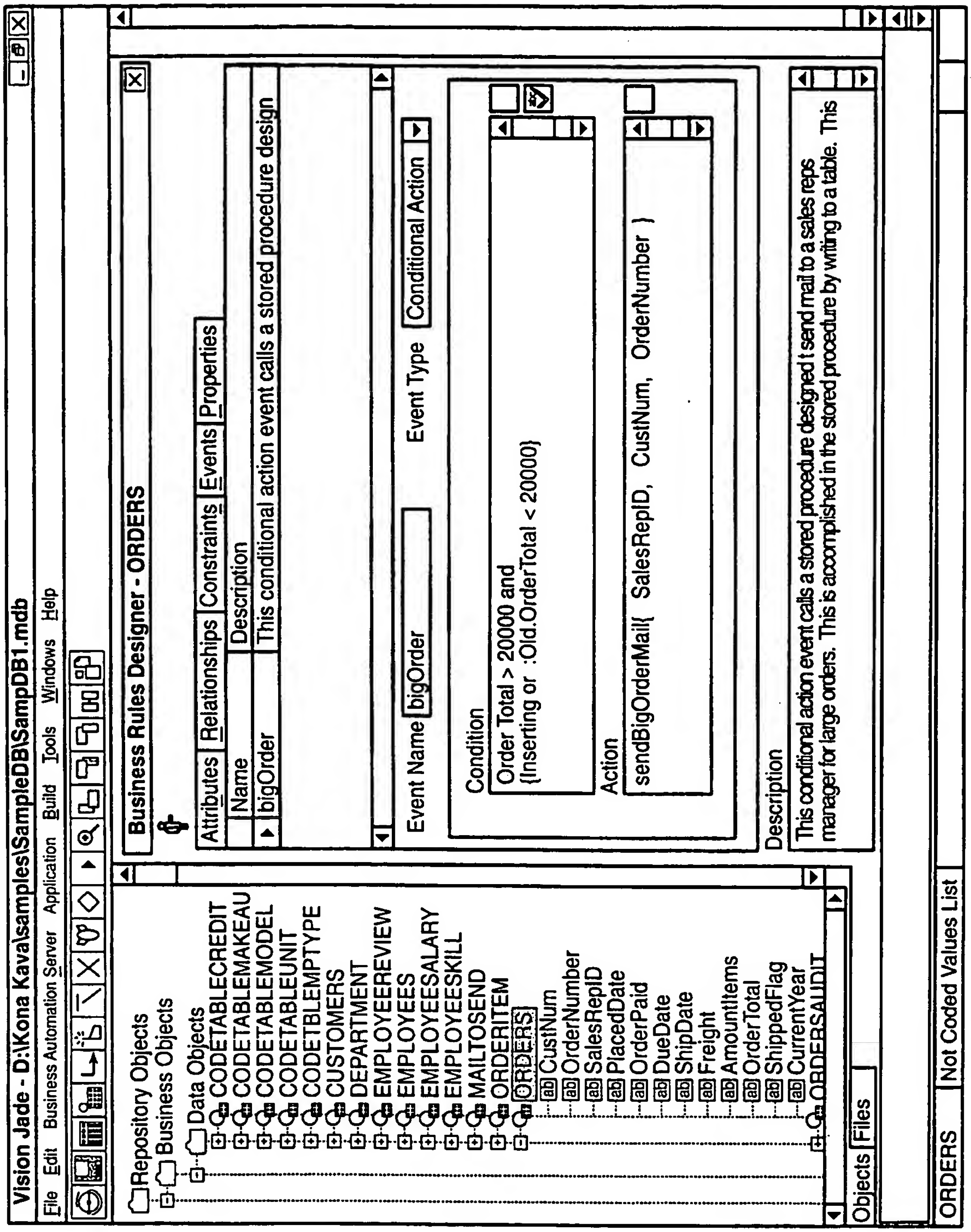

Vision Jade - D:\Kona Kava\samples\SampleDB\SampDB1.mdb
File Edit Business Automation Server Application Build Tools Windows Help
Repository Objects
Business Objects
Data Objects
CODETABLECREDIT
CODETABLEMAKEAU
CODETABLEMODEL
CODETABLEUNIT
CODETBLEMPTYPE
CUSTOMERS
DEPARTMENT
EMPLOYEEREVIEW
EMPLOYEES
EMPLOYEESALARY
EMPLOYEESKILL
MAILTOSEND
ORDERITEM
ORDERS
CustNum
OrderNumber
SalesRepID
PlacedDate
OrderPaid
DueDate
ShipDate
Freight
AmountItems
OrderTotal
ShippedFlag
CurrentYear
Objects
Files
Business Rules Designer - ORDERS
Attributes
Relationships
Constraints
Events
Properties
Name
Description
bigOrder
This conditional action event calls a stored procedure design
Event Name
bigOrder
Event Type
Conditional Action
Condition
Order Total > 20000 and
(Inserting or :Old.OrderTotal < 20000)
Action
sendBigOrderMail(SalesRepID, CustNum, OrderNumber)
Description
This conditional action event calls a stored procedure designed t send mail to a sales reps manager for large orders. This is accomplished in the stored procedure by writing to a table. This
ORDERS
Not Coded Values List

FIG._6

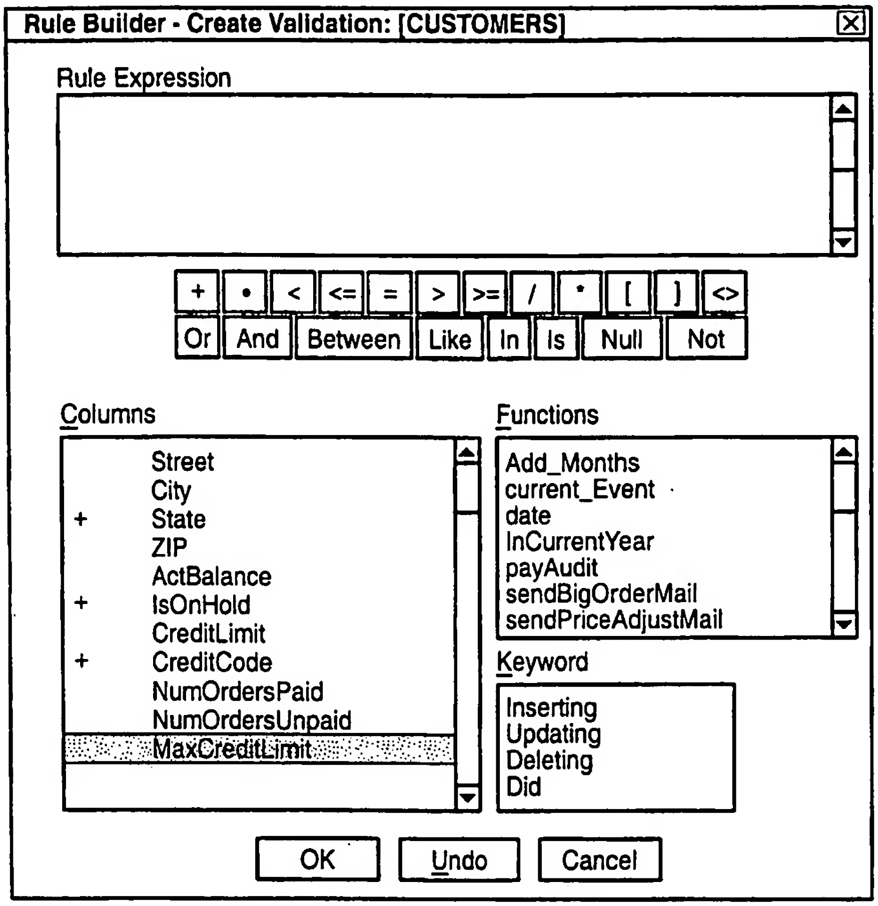

Rule Builder - Create Validation: [CUSTOMERS]
Rule Expression
+
•
<
<=
=
>
>=
/
*
[
]
<>
Or
And
Between
Like
In
Is
Null
Not
Columns
Street
City
+ State
ZIP
ActBalance
+ IsOnHold
CreditLimit
+ CreditCode
NumOrdersPaid
NumOrdersUnpaid
MaxCreditLimit
Functions
Add_Months
current_Event
date
InCurrentYear
payAudit
sendBigOrderMail
sendPriceAdjustMail
Keyword
Inserting
Updating
Deleting
Did
OK
Undo
Cancel

FIG._7

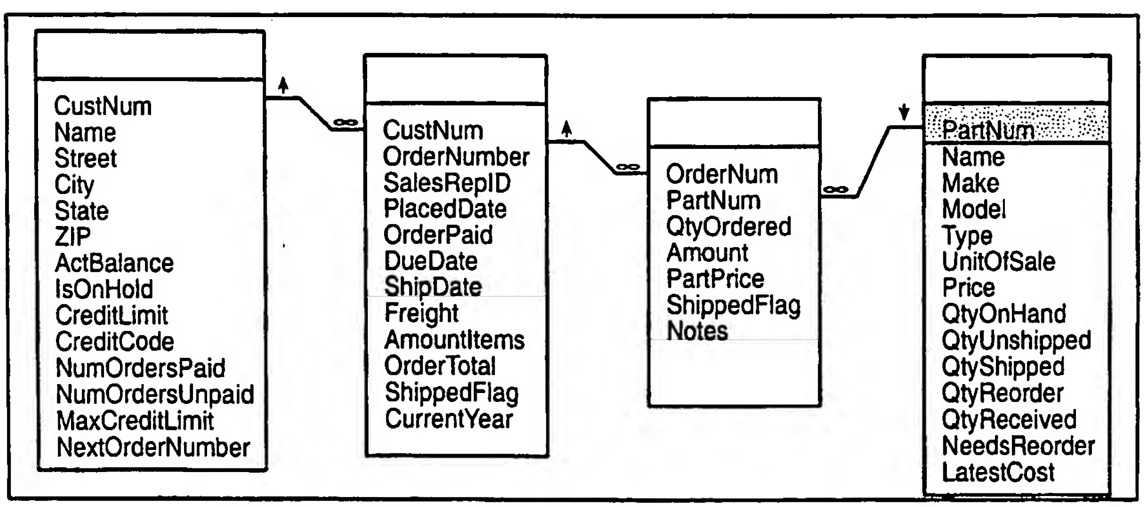

CustNum
Name
Street
City
State
ZIP
ActBalance
IsOnHold
CreditLimit
CreditCode
NumOrdersPaid
NumOrdersUnpaid
MaxCreditLimit
NextOrderNumber
CustNum
OrderNumber
SalesRepID
PlacedDate
OrderPaid
DueDate
ShipDate
Freight
AmountItems
OrderTotal
ShippedFlag
CurrentYear
OrderNum
PartNum
QtyOrdered
Amount
PartPrice
ShippedFlag
Notes
PartNum
Name
Make
Model
Type
UnitOfSale
Price
QtyOnHand
QtyUnshipped
QtyShipped
QtyReorder
QtyReceived
NeedsReorder
LatestCost

FIG._8


DECLARATIVE BUSINESS RULES REPORT
RULES MAPPING
RULES ORDERING
RULES OPTIMIZATION
RULES TARGETING

FIG._9

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US00/22318

A. CLASSIFICATION OF SUBJECT MATTER
IPC(7) :G06F 17/60
US CL :705/ 7, 8, 9
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : 705/ 7, 8, 9

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
WEST 2.0, CAS ONLINE, DIALOG, IEEE

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5,191,522 A (BOSCO et al) 02 March 1993, see entire document.	1-19
A,P	US 5,960,200 A (EAGER et al) 28 September 1999, see entire document.	1-19
A	US 6,016,477 A (EHNEBUSKE et al) 18 January 2000, see entire document.	1-19

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
31 OCTOBER 2000	06 DEC 2000

Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231
Facsimile No. (703) 305-3230

Authorized officer
TARIQ R. HAFIZ James R. Matthews
Telephone No. (703) 305-9643